

Majord'Home

Documentation technique

Dans un premier temps, nous ferons un rappel du fonctionnel de l'application, ensuite, nous commenterons le logiciel grâce aux différents diagrammes qui permettront d'illustrer notre programme d'un point de vue global et détaillé, puis nous expliquerons en détails le protocole de communication ainsi que les fichiers de données. Enfin, nous détaillerons le rôle des principales classes à travers un dictionnaire des méthodes.

13 aout 2012

K.BRETÉCHER - K.BERLAT - R.ESCUDIER - T.GARET - P.MANOT - L.CHAUVEL

Groupe :	berlat_k bretec_k chauve_l garet_t escudi_r manot_p
Nom du projet :	**MAJORD'HOME**
Type de document :	Documentation technique
Version :	0.6
Référence :	2013_TD4_FR_majordhome
Statut du document :	En cours

Date	Version	Description	Auteur
25/01/2012	0.1	Création de toutes les parties	Karim Berlat
26/01/2012	0.2	Ajout du dictionnaire des méthodes serveur et client desktop	Karim Berlat
09/03/2012	0.3	Mise à jour du plan	Karim Berlat
09/03/2012	0.4	Ajout des sections références et bugs techniques	Karim Berlat
30/04/2012	0.5	Modification de toutes les sections du document en prenant en compte les commentaires du DA2	Karim Berlat
12/05/2012	0.6	Correction de toutes les sections	Karim Berlat

SOMMAIRE

1. Nature du document

1.1 Objectif du document

La documentation technique a pour but d'expliquer les différentes parties implémentées et fonctionnelles telles que:

- le protocole de communication ;
- le stockage des données ;
- le code produit à travers un dictionnaire des méthodes.

1.2 Périmètre

1.2.1 Ce que comprend ce document

- le contexte du projet ;
- un rappel sur Majord'Home ;
- le détail du fonctionnement client/serveur;
- le détail du protocole de communication ;
- le détail du stockage des vues ;
- le détail sur la structure des données ;
- la description des classes et méthodes implémentées ;

1.2.2 Ce que ne comprend pas ce document

- les objectifs commerciaux ;
- les informations sur d'éventuels sponsors et partenariats ;
- le manuel d'utilisation.

2. Introduction

2.1 Contexte

Majord'Home est un Epitech Innovative Project (EIP) qui regroupe six membres : Kévin BRETÉCHER, Karim BERLAT, Romain ESCUDIER, Thomas GARET, Paul-Baptiste MANOT et Ludovic CHAUVEL. Un EIP est un projet innovant au sein d'Epitech qui se déroule sur une période de trois ans et qui finalise le cursus EPITECH. Au terme de cette période, le groupe présentera Majord'Home au forum des innovations 2013.

Ce projet s'inscrit dans le domaine de la domotique et a pour but de piloter différentes parties d'une maison (portail, lumière, HIFI, alarmes, climatisation…). La domotique est l'ensemble des techniques de l'électronique, de physique du bâtiment, d'automatismes, de l'informatique et des télécommunications utilisées dans les bâtiments. La domotique vise à apporter des fonctions de confort (optimisation de l'éclairage, du chauffage), de gestion d'énergie (programmation), de sécurité (comme les alarmes) et de communication (comme les commandes à distance ou l'émission de signaux destinés à l'utilisateur) que l'on peut retrouver dans les maisons, les hôtels, les lieux publics...

2.2 Historique

L'idée de développer Majord'Home a grandi au sein d'un groupe de six étudiants, passionnés par l'informatique et les technologies innovantes.

Le nom « Majord'Home » rappelle le mot « majordome » et « home » est un mot anglais se traduisant par « maison ». Nous avons trouvé que le jeu de mots était intéressant.

2.3 Rappel sur notre EIP

2.3.1 Objectif de l'EIP

Afin de mener notre projet à bien, nous nous sommes imposés comme objectif principal:

- la réalisation complète d'une solution client-serveur.

Comme objectifs secondaires:

- la réalisation de différents modules qui seront basés sur le protocole EIB/KNX. Ce protocole est respecté par la majorité des solutions du marché de la domotique, ce qui confirme notre choix ;
- la commercialisation de notre solution.

2.3.2 Sujet de notre EIP

Majord'Home est une solution qui s'inscrit dans la domotique, il pourra être piloté par une interface graphique, une interface vocale et une interface mobile.

L'utilisateur peut utiliser différents types de solution clients comme un Smartphone (ou une tablette électronique) sous Androïd ou Windows Phone, un ordinateur sous Windows ou Linux.

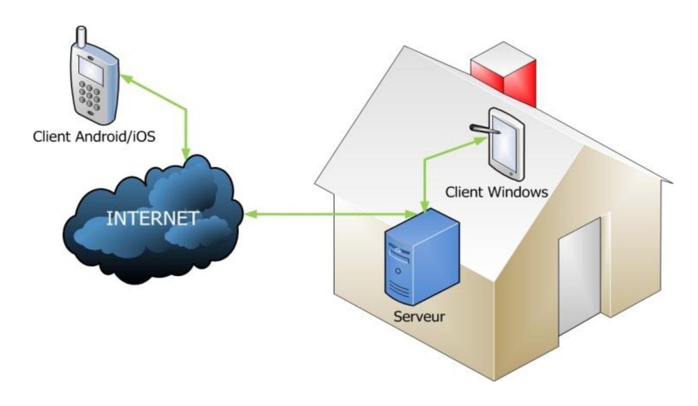


FIGURE 1 SCHEMA GENERAL

Depuis l'emplacement où est installé le système, il sera également possible de communiquer avec Majord'Home via une interface vocale. Ainsi Majord'Home sera à même d'intégrer différents modules tels que:

- la gestion de périphériques: HIFI, TV et informatique ;
- la gestion de lumière, climatisation et chauffage ;
- la gestion de portes et portails ;
- et la gestion d'alarmes et caméras de surveillance.

Majord'Home est innovant puisqu'il intégrera une interface vocale et une interface mobile sous Androïd et Windows Phone. Certes il existe des solutions à contrôle vocale mais celles-ci ne sont pas compatibles avec des systèmes mobiles tels qu'Androïd.

Nous avons choisi de travailler sur un projet en rapport avec la domotique du fait que nous sommes convaincus que ce genre de solution deviendra indispensable.

3. Cas d'utilisation

Dans le diagramme des cas d'utilisations, il y a deux parties: une vocale et une tactile. Toute les actions se rapportant au contrôle via l'interface tactile forment la liste exhaustive des fonctionnalités gérées par le client Windows/Linux, les seules fonctionnalités non présentes dans le client mobile représentent toutes les actions se rapportant à la gestion des profils (créer un mode, paramétrer un mode, supprimer un mode).

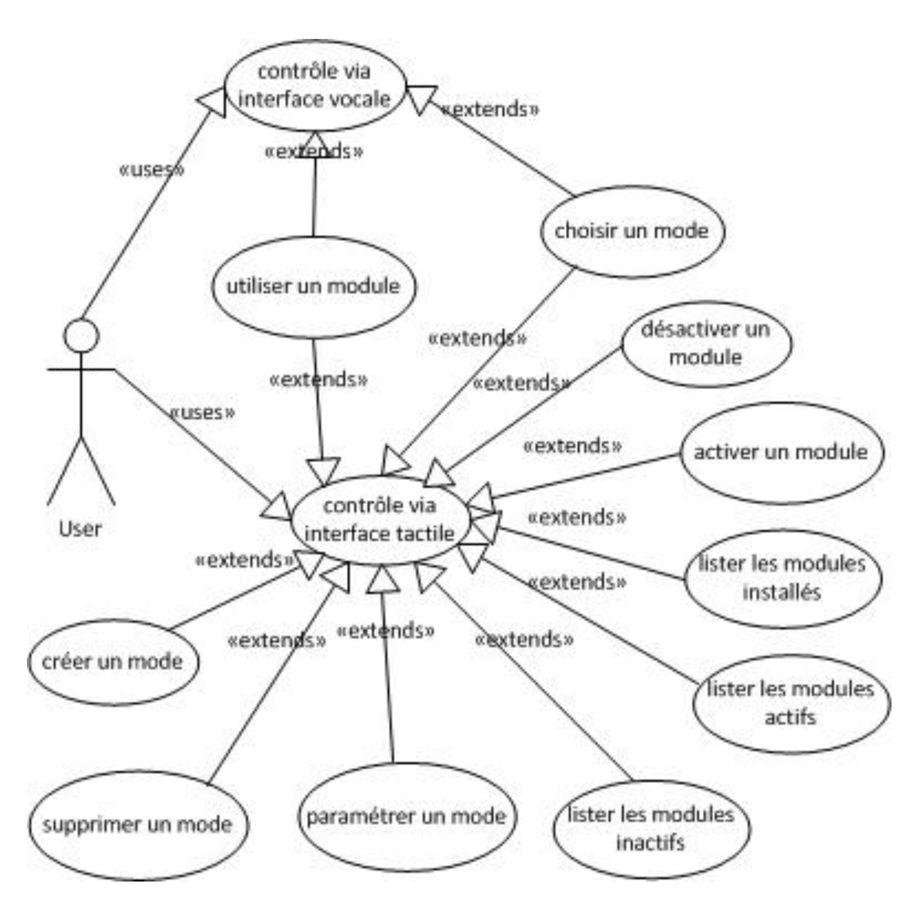


Figure 2 : Cas d'utilisation

3.1 Activer un module

Pré condition:

- se situer sur l'interface de gestion des modules de l'application.

Interactions:

- le client doit sélectionner un module dans la liste, et choisir de l'activer.

Résultats observables:

- l'état du module passera à actif dans le tableau de contrôle.

3.2 Désactiver un module

Pré condition:

- se situer sur l'interface de gestion des modules de l'application.

Interactions:

- le client doit sélectionner un module dans la liste, et choisir de le désactiver.

Résultats observables:

- l'état du module passera à inactif dans le tableau de contrôle.

3.3 Lister les modules actifs

Pré condition:

- se situer sur l'interface de gestion des modules de l'application.

Interactions:

- le client doit sélectionner le filtre « modules actifs » se trouvant sur la page.

Résultats observables:

- seuls les modules actifs seront listés.

3.4 Lister les modules inactifs

Pré condition:

- se situer sur l'interface de gestion des modules de l'application.

Interactions:

- le client doit sélectionner le filtre « modules inactifs » se trouvant sur la page.

Résultats observables:

- seuls les modules inactifs seront listés.

3.5 Lister les modules installés

Pré condition:

- se situer sur l'interface de gestion des modules de l'application.

Interactions:

- le client doit sélectionner le filtre « modules installés » se trouvant sur la page.

Résultats observables:

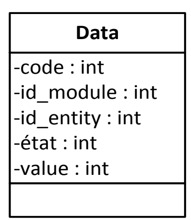
- seuls les modules installés seront listés.

4. Protocole

4.1 Structure des données

Coté client:

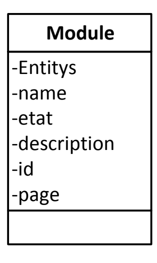
- une structure Data pour la communication client/serveur ;



- une structure Entity pour chaque entité ;

Entity
-id : int -etat : int -nom : string -emplacement : string -type : int -button -bar

- une structure module pour le stockage des données d'un module.



Coté serveur:

- une structure Entity décrite ci-dessus ;
- une structure Data pour la communication client/serveur également décrite ci-dessus ;
- une classe module contenant les données pour chaque module.

4.2 Le protocole de communication

Lors de la connexion, le client envoi une demande de fichier au serveur. Pour la communication client/serveur, nous avons mis en place un protocole qui consiste à envoyer toutes les informations nécessaires via une séquence de cinq Int.

Lors de l'envoi de la vue, la séquence est décrite de la manière suivante:

- Int 1 correspond au type d'échange (envoi de fichier, envoi de commande…) ;
- Int 2 correspond à la taille du fichier à envoyer ;
- Les trois derniers Int ne sont pas utilisés pour l'envoi de la vue, ils sont donc mis à zéro ;
- Contenu du fichier.

Lors de l'envoi d'une action, la séquence est décrite de la manière suivante:

- Int 1 correspond au type d'échange (envoi de fichier, envoi de commande…) ;
- Int 2 correspond l'ID du module concerné ;
- Int 3 correspond à l'ID de l'entité ;
- Int 4 l'état de l'entité (actif, inactif, ouvert, fermé…) ;
- Int 5 correspond à la valeur appliquée.

4.3 Stockage des vues dynamiques

Le client de Majord'Home est composé d'une interface statique partiellement dynamique. Pour la partie dynamique, les différentes vues sont générées par le client à partir d'un fichier au format XML.

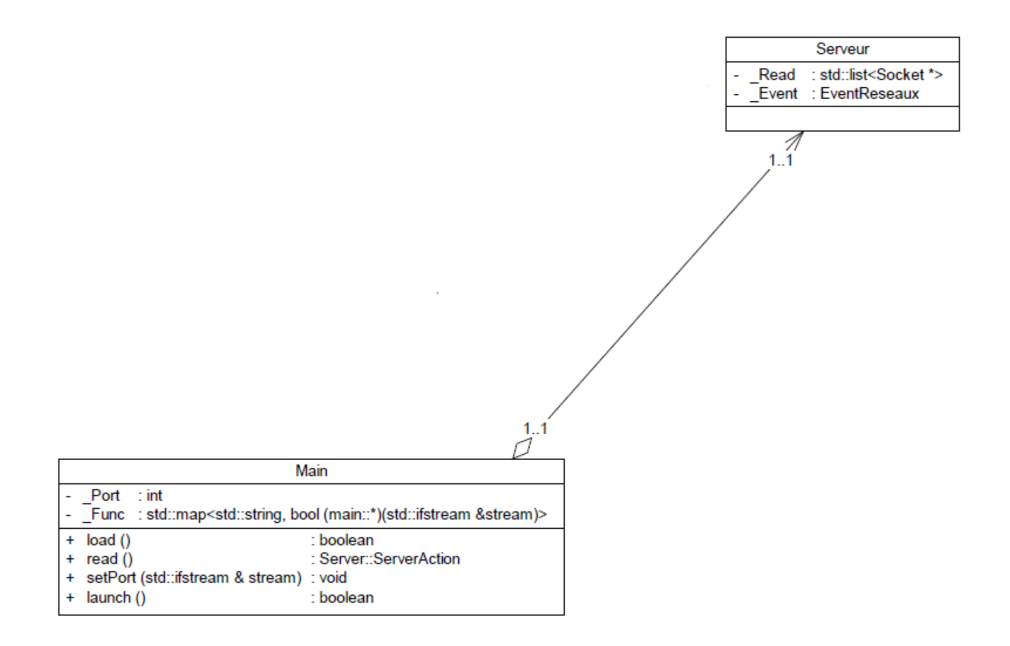
Voici un exemple de fichier XML pour la vue du module «Lumière »:

```
<vue>
  <page periph="lumiere" id="0">
    <entity emplacement="maison 1" id="1" label="on/off" value="90" type="3" etat="1">lumiere 1</entity>
    <entity emplacement="maison 1" id="2" label="on/off" value="30" type="3" etat="1">lumiere 2</entity>
    <entity emplacement="maison 1" id="3" label="on/off" value="30" type="3" etat="1">lumiere 3</entity>
  </page>
</vue>
```

5. Logiciel

5.1 Le serveur

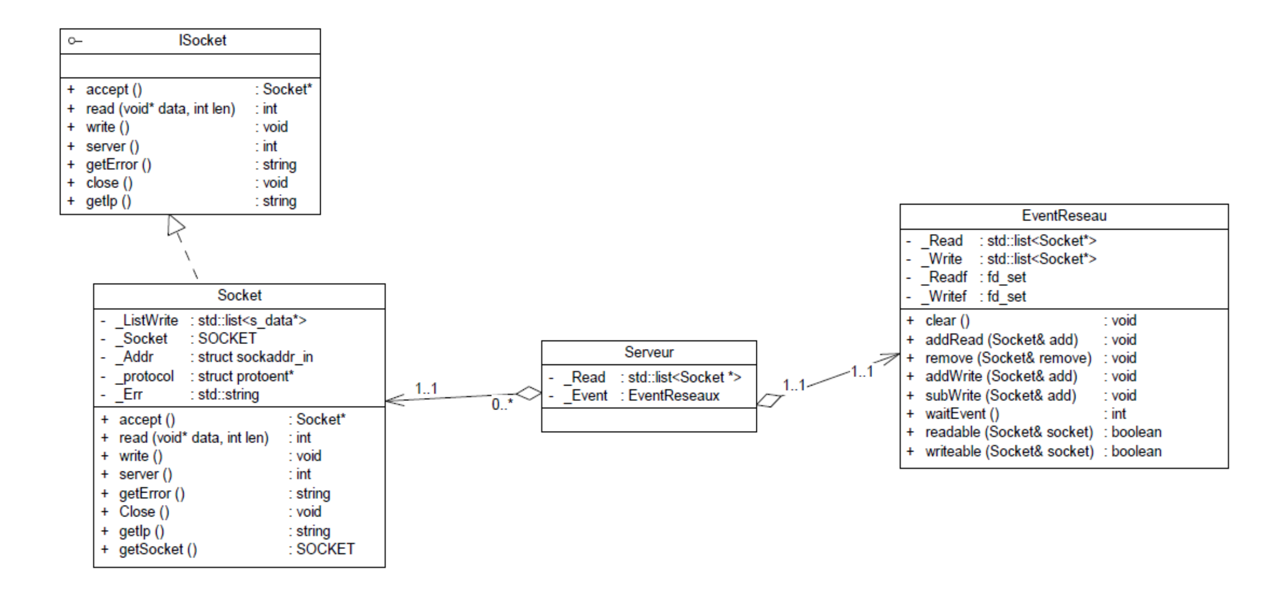
Concernant le serveur, la classe principale est « Main », elle permet de stocker le serveur. Elle contient la classe « serveur », qui permet d'assurer la liaison entre les différentes parties du serveur.



Cette classe « Serveur » se divise en plusieurs sous-classes:

- la classe « socket » correspond à la socket utilisée pour la communication avec le client. Les méthodes les plus importantes de cette classe sont les suivantes:
    - server() pour créer le serveur à partir de cette socket ;
    - accept() pour accepter une nouvelle connexion sur cette socket ;
    - read() pour recevoir les données ;
    - write() pour envoyer des données ;
    - close() pour fermer la connexion ou le serveur suivant le type de socket.

- la classe « EventReseaux » qui s'occupe de traiter les évènements réseaux. Chaque socket est ajoutée dans cette classe avec la méthode « addRead() » pour la lecture ou « addWrite() » pour l'écriture.

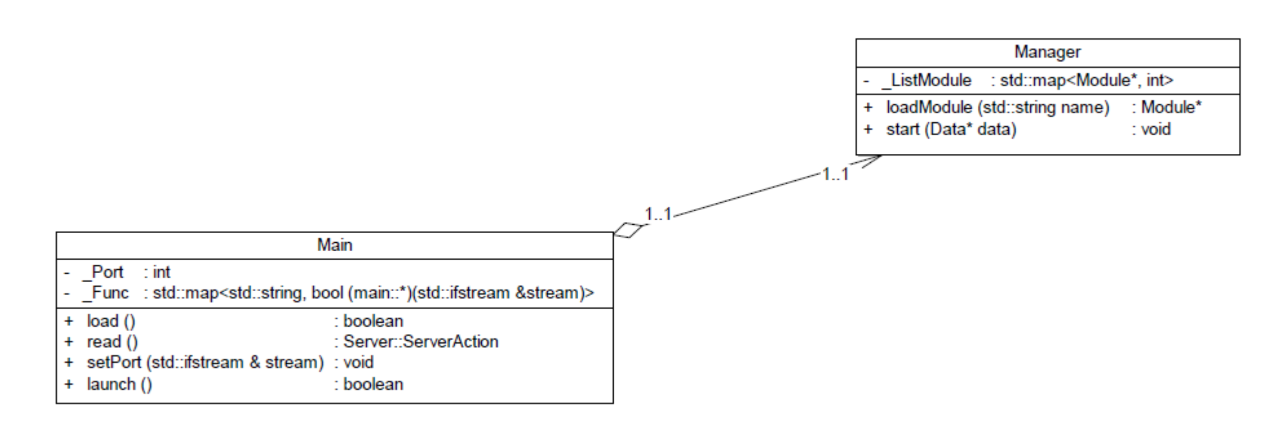


- la méthode « waitEvent » met en pause le serveur jusqu'à ce qu'un évènement se produise sur une des sockets surveillées.

- les méthodes « readable » et « writeable » permettent de savoir si on peut lire ou écrire sur une socket.

La boucle principale se situe au niveau de la classe « Serveur », elle permet de lancer toutes les routines de surveillance du réseau.

Une fois un paquet reçu, la classe « Serveur » remonte les données à la classe « main ». Celle-ci traduit les données pour les reconduire dans la classe « Manager ».



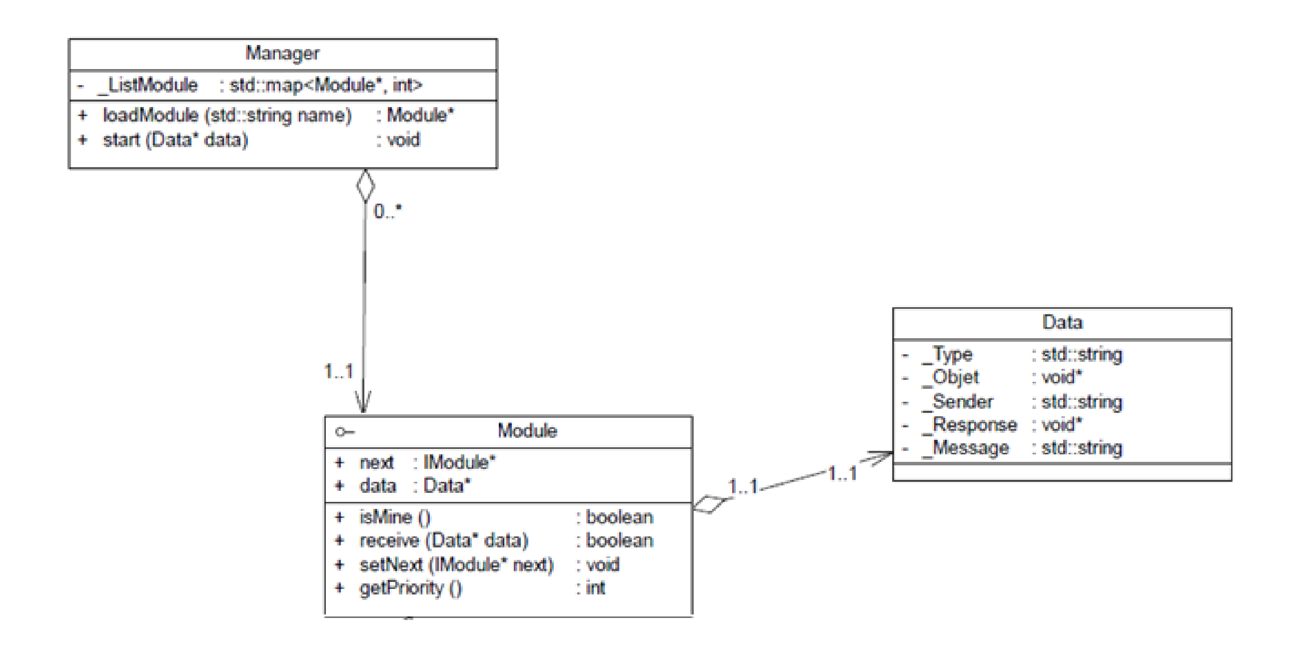
La classe « Manager » contient tous les modules (chargés de manière dynamique dans le dossier module de l'exécutable) nécessaires pour le bon fonctionnement du serveur ; à savoir un module pour les lumières, un autre pour les volets …

La méthode « loadModule » charge un module en mémoire à partir de son nom.

La méthode « execute » transmet les données au premier module de la liste.

La classe implémentant les modules s'appelle « Module ». Les différents modules sont instanciés de manière dynamique, le code est donc différent pour chaque module, d'où l'utilisation d'une classe « Module » composée de méthodes virtuelles. La méthode principale de cette classe est « receive » qui reçoit les données transmises par la class « Manager ». Une fois les données interprétées par ce module, elles sont transmises au module suivant via la méthode « receive » du prochain module.

Pour terminer, la classe « Data » contient les paquets reçus et émis par le serveur. Pour la génération du XML de communication entre le serveur et le client, chaque module possède une méthode (Vue) qui ajoute dans un fichier les données le concernant.



5.2 Le client

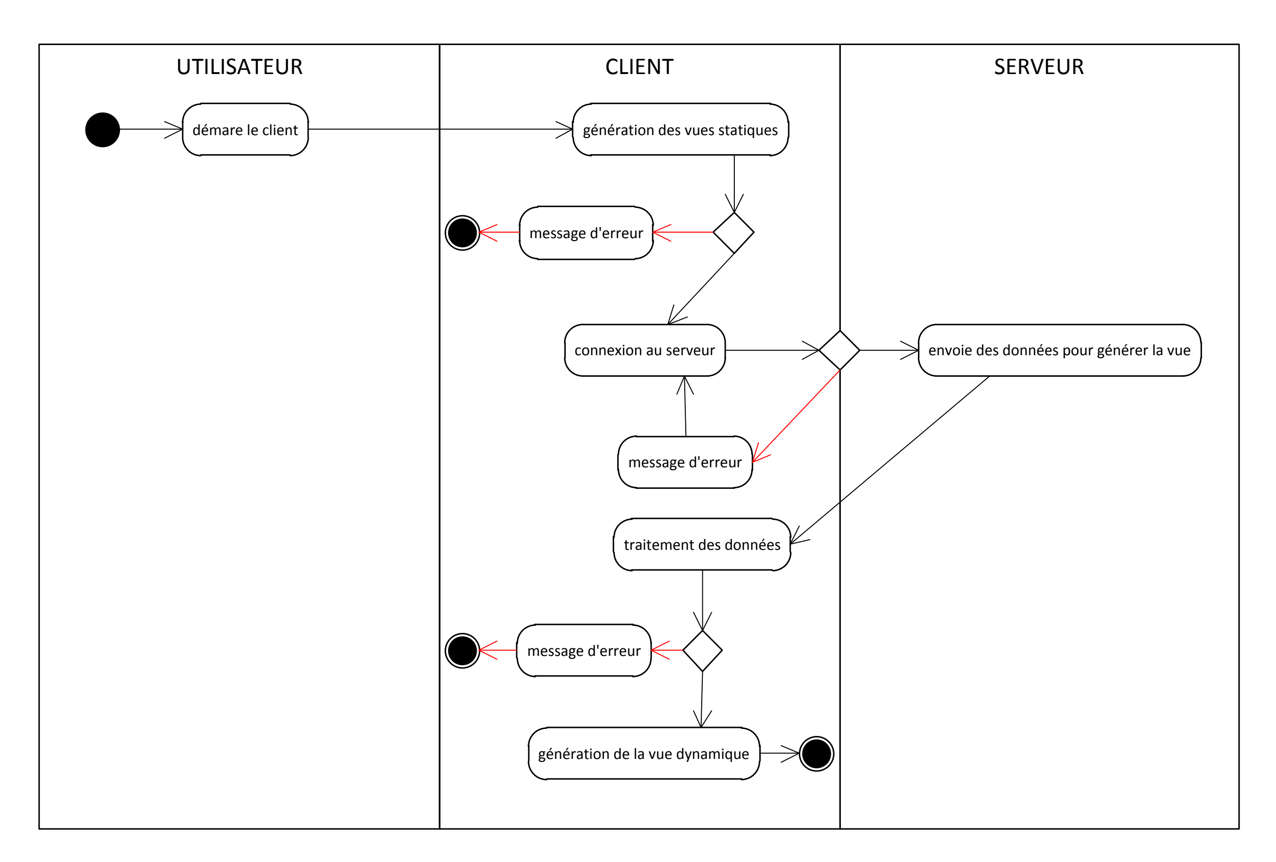


FIGURE 3: DIAGRAMME D'ACTIVITES

Le client reçoit un fichier de configuration au format XML via le réseau. Ce fichier de configuration contient toutes les informations nécessaires à la génération dynamique de la vue des clients (Androïd/Windows Phone et Windows/Linux) et permet d'activer les parties du contrôleur nécessaires à son fonctionnement.

Nous avons défini une classe «MainWindow», qui est composée de plusieurs méthodes :

- « MainWindow::Parse » dont le rôle est de parser le fichier pour générer la partie dynamique des pages ;
- « MainWindow::Design » qui crée la partie statique des pages ;
- « MainWindow::createPageModule » qui crée les onglets pour les modules disponibles, c'est-à-dire que les pages sont générées si le module est existant.

Cela permet, d'ajouter des modules sans avoir à reprogrammer la vue. C'est la raison pour laquelle nous avons privilégié un fonctionnement dynamique et modulaire.

Après avoir reçu, de la part du serveur, le fichier XML à parser, la méthode « MainWindow::loadVue » est appelée. Cette méthode a pour but de vérifier la bonne réception du fichier et d'appeler la méthode de parsage, puis d'appeler « MainWindow::createPageModule » pour générer la vue dynamique, créer les différentes pages agrémentées de leurs entités et afficher les différentes pages.

Cette classe « MainWindows » est la brique centrale pour chaque client, en effet elle contient toutes les classes nécessaires au fonctionnement de ce dernier.

Nous utilisons une classe «Page» pour chaque page de la vue. Chaque page est constituée d'une liste d'entités qui représente la liste des objets présents dans cette dernière.

Chaque élément composant une page est de type « Entity ». Cette structure contient toutes les informations nécessaires pour chaque élément.

6. Dictionnaire des méthodes

6.1 Serveur

Class::méthode	Description
Void Main::sendVue(Socket *Client)	Envoi le fichier XML au client
Void Main::traite(Data &data, Socket *Client)	Transmet les données passées en paramètres au manager des modules
Server::ServerAction Main::read(Socket *Client)	Récupère les données sur la socket du client
Bool Main::LoadModule(TiXmlNode *element)	Charge un module, contenu dans le fichier XML de configuration
Bool Main::setPort(TiXmlNode *element)	Défini le port à utiliser pour le serveur, contenu dans le fichier XML de configuration
Bool Main::Parse(TiXmlNode *element)	Lance le parsage du fichier XML de configuration
Bool Main::Load()	Charge le serveur avec les paramètres définis dans le fichier de configuration
Bool Main::Launch()	Demande au serveur de démarrer
Server::ServerAction lire(Socket *client, void *data)	Lit les données provenant d'un client
Void EventReseaux::clear()	Efface toutes les sockets contenues dans la liste de lecture et d'écriture
Void EventReseaux::addRead(Socket &add)	Ajoute une socket à surveiller en lecture
Void EventReseaux::remove(Socket &remove)	Ferme la connexion avec une socket
Void EventReseaux::addWrite(Socket &add)	Ajoute une socket à surveiller en écriture
Void EventReseaux::subWrite(Socket &sub)	Supprime une socket à surveiller en lecture
Int EventReseaux::waitEvent()	Met le serveur en pause, attendant un événement réseau
Bool EventReseaux::Readable(Socket &socket)	Permet de tester si un read ne bloquera pas sur une socket
Bool EventReseaux::Writeable(Socket &socket)	Permet de tester si un write ne bloquera pas sur une socket

Void Manager::execute(Data *data)	Transmet les données passées en paramètre au premier module de la liste
Bool Manager::loadModule(std::string name)	Charge le module « name »
Void Manager::createXML()	Crée le XML à envoyer au client
Bool Module::Receive(Data *data)	Reçoit les données passées par le manager en paramètre et les traite
Void Module::setNext(Module *next)	Défini le module suivant de la liste
std::string Module::getType()	Retourne le type de module (lumière, volet, …)
int Module::getId()	Retourne l'ID du module
Void Server::setReadFunction(ServerAction (*func)(Socket *client, void *data))	Permet de définir la fonction à appeler lorsqu'une socket est prête à être lue
Void Server::setPort(const int &port)	Défini le port du serveur
const int &Server::getPort()	Retourne le port du serveur
const bool Server::Launch(void *data)	Lance le serveur
std::string Server::getErrorMessage(Server::ServerError error)	Retourne le message d'erreur correspondant au paramètre
Server::ServerError Server::getError()	Retourne la dernière erreur survenue sur le serveur
Void Server::WriteLog(std::string message)	Ecrit dans le fichier de log le message passé en paramètre
Void Server::Err(ServerError error)	Enregistre l'erreur survenue et l'affiche sur la sortie standard
Void Server::Err(std::string function, char *message)	Affiche sur la sortie standard la fonction « function » puis le message d'erreur
Void Server::Err(std::string message)	Affiche « message » sur la sortie standard
Void Server::subSocket(Socket *socket)	Supprime une socket du serveur
Void Server::addSocket(Socket *socket)	Ajoute une socket au serveur
Void Server::myAccept()	Accepte une connexion d'un client et rajoute sa socket dans le serveur
Void Server::WriteClient(Socket *socket, const void *data, int len)	Envoi les données passées en paramètre à travers la socket client

Void Server::WriteClient(Socket *socket, void *data, int len)	Surcharge de WriteClient pour une donnée non constante
Void Server::Loop()	Boucle principale du serveur
Void Socket::Close()	Ferme la socket
Int Socket::getSocket()	Récupère l'ID de la socket (Unix)
SOCKET Socket::getSocket()	Récupère la socket (Windows)
Socket *Socket::accept()	Pour une socket serveur, accepte une connexion entrante
Int Socket::read(void *data, int len)	Lit sur la socket les données disponibles
Void Socket::write(void *data, int len)	Ecrit sur la socket les données passées en paramètre
Bool Socket::server(int port)	Démarre un serveur sur la socket avec le port passé en paramètre
std::string Socket::getError()	Récupère la dernière erreur survenue pour cette socket
const std::string Socket::getIp()	Récupère l'IP de la socket (pour une socket serveur: 127.0.0.1, pour une autre: IP du client)
void Server::setDeleteFunction(void (*func)(Socket *client, void *data))	Défini la fonction à appeler lors de la suppression d'une socket

6.2 Client Desktop

Class::méthode	Description
Void MainWindow::createPageModule()	Crée la page dynamique pour les modules
void MainWindow::Close()	Ferme la fenêtre principale
Void MainWindow::ButtonClicked(Entity *entity)	Fonction appelée lors du clic sur un bouton d'une entité
void MainWindow::DeactivateModule()	Désactive un module
void MainWindow::Launch()	Lance l'application
void MainWindow::LoadVue()	Charge la vue a partir du XML
Void MainWindow::ParseModule(TiXmlNode *node)	Parse un module a partir du XML

void MainWindow::Read()	Demande au réseau de lire les données provenant du serveur
void MainWindow::ModifyButton(Entity *entity)	Met à jour le bouton correspondant au fichier
void MainWindow::createPageModule(struct Module *module)	Crée une page dynamique concernant le module passé en paramètre
Void MainWindow::createLineModuleTable(bool etat, std::string Description, std::string Name, int l, QStandardItemModel *model)	Crée une ligne dans le tableau de la page principale
QStandardItemModel * MainWindow::CreateModuleTable()	Crée la structure du tableau à afficher sur la page principale
void MainWindow::setModuleTableModel (QStandardItemModel *model)	Défini le model a utiliser pour le tableau de la page principale
void MainWindow::Design()	Créer le design de la fenêtre pour la partie statique
void Network::setIp(std::string pi)	Défini l'IP à utiliser pour la connexion au serveur
void Network::setPort(int port)	Défini le port à utiliser pour la connexion avec le serveur
bool Network::Connect()	Connecte le client au serveur
Void Network::Write(void *data, int len)	Ecrit les données sur la socket du serveur
void Network::Read(void *data, int len)	Lit les données en provenance du serveur
void Network::setReadFunc(void (MainWindow::*func)())	Défini la fonction à utiliser pour la lecture de données
void Network::ErrorSocket(QAbstractSocket::SocketError erreur)	Récupère la dernière erreur sur la socket
void Network::Read()	Lit les données reçues du serveur

7. Bugs Techniques

Ci-dessous la liste des bugs connus. Veuillez nous excuser pour la gêne occasionnée.

- Crash possible du client Desktop Windows lorsqu'il tente d'établir la connexion avec le serveur.

8. Références

Les documentations Majord'Home sont disponibles sur le site officiel Majord'Home:

- www.major-dhome.com ;
- www.major-dhome.fr.

Documentation Doxygen : http://eip.epitech.eu/2013/majordhome/Doxygen/index.html